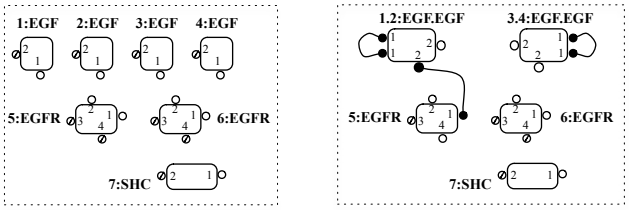
A Biochemical Calculus Based on Strategic Graph Rewriting

Oana Andrei[1] and Hélène Kirchner[2]

[1] INRIA Nancy Grand-Est & LORIA, France
[2] INRIA Bordeaux Sud-Ouest, France
First.Last@loria.fr

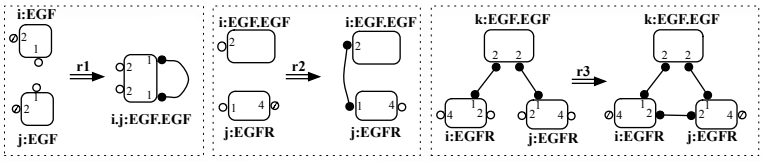
When modeling interactions between molecules or proteins, the behaviour of a protein is given by its functional domains that determine which other protein it can bind to or interact with and these domains are usually abstracted as sites that can be bound or free, visible or hidden. Hence a protein is characterized by the collection of interaction sites on its surface and proteins can bind to each other forming molecular complexes. Based on such structures, we considered port graphs [1] which are graphs with ports and with multiple edges and loops attached to ports of nodes. Molecular complexes are port graphs where each port is connected to at most one other port. Such restricted port graphs are called *molecular graphs* and their ports are called *sites*.

We illustrate below a molecular graph G representing the initial state of the system modeling a fragment of the EGFR signaling cascade [12, 14]. The protagonists of this model are three types of proteins: the signal *EGF*, the receptor *EGFR*, and the adapter *SHC*. The molecular graph G' represents a state of the system where two signal proteins are already bound forming a dimer binding in turn a receptor. A node is graphically represent as a box with an unique identifier and a name placed outside the box. A site is represented as a filled, empty, or slashed circle on the surface of the box if its state is respectively bound, free, or hidden.



A *molecular graph rewrite rule* $L \Rightarrow R$ is a port graph consisting of two molecular graphs L and R called, as usual, the left- and right-hand side respectively, and one special node $\Rightarrow$, called the *arrow node* with ports connected to the sites of L and R such that it embeds the correspondence between elements of L and elements of R. We represent graphically the edges incident to the arrow node only if the correspondence is ambiguous. In consequence, port graphs represent a unifying structure for representing both molecular complexes and the reaction patterns between them.

The five reaction patterns for the EGFR signalling cascade fragment specify the followings: **(r1)** two signaling proteins form a dimer represented as a single node; **(r2)** an EGF dimer and a receptor bind on free sites; **(r3)** two receptors activated by the same EGF dimer bind creating an active dimer RTK; **(r4)** an active dimer RTK activates itself by attaching phosphate groups; **(r5)** an activated RTK binds to an adapter protein activating it as well. They are easily expressible using molecular graph rewrite rules. The first three rules have the following graphical representation:



Let $r : L \Rightarrow R$ be a molecular graph rewrite rule and G_1 a molecular graph such that there is an injective graph morphism g from L to G_1. By replacing the subgraph $g(L)$ for $g(R)$ and connecting it appropriately in the context, we obtain a molecular graph G_2 which represents a result of *one-step rewriting* of G_1 using the rule r, written $G_1 \rightarrow_r G_2$.[3] The formal definition of port graph rewriting is given in [1]. An example of rewriting in the EGFR fragment is the molecular graph G' above obtained from the initial molecular graph G by rewriting it using twice the rule **r1** and once the rule **r2**.

The chemical computation metaphor emerged as a computation paradigm over the last three decades. This metaphor describes computation in terms of a chemical solution in which molecules representing data freely interact according to reaction rules. Chemical solutions are represented by multisets and the computation proceeds by rewritings, which consume and produce new elements according to conditions and transformation rules. The chemical metaphor was proposed as a computational paradigm in the Γ language in [7], then used as a basis for defining the CHemical Abstract Machine (CHAM) [8], and later it was extended to the γ-calculus and HOCL in [5, 6] for modeling self-organizing systems in particular.

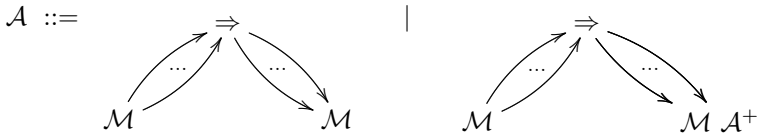
We extend the chemical model with high-level features by considering a port graph structure for the data and the computation rules. The result is a port graph rewriting calculus with higher-order capabilities, called the ρ_{pg}-calculus. The first citizens of the ρ_{pg}-calculus are port graphs, port graph rewrite rules, and rule application. This calculus generalizes the rewriting calculus [11] and the term graph rewriting calculus [9]. The ρ_{pg}-calculus also generalizes the λ-calculus and the γ-calculus through a more powerful abstraction power that considers for matching not only a variable but a port graph with variables.

The ρ_{pg}-calculus is a suitable formalism for modeling systems whose states are port graphs and whose transitions are reductions obtained by applying port graph rewrite rules. Due to the intrinsic parallel nature of rewriting on disjoint

[3] There can be different such morphisms g from L to G_1 leading to different rewrites.

redexes and decentralized rule application, we thus model a kind of *Brownian motion*, a basic principle in the chemical paradigm. In the following we present the main features of the syntax and the semantics of the calculus from a biochemical modeling point of view.

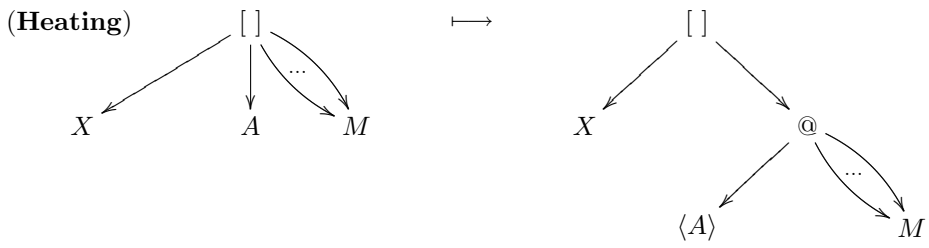
Let $\mathcal{M}$ denote the class of molecular graphs modeling systems states. We denote by $\mathcal{A}$ the class of *abstractions* which are port graph rewrite rules whose left-hand sides are molecular graphs and whose right-hand sides may include other abstractions as well. Both sides are connected through the arrow node. Let $\mathcal{A}^+$ denote a non-empty set of abstractions. Then the abstractions are graphically defined as follows:



The second type of abstraction enriches the expressivity of the calculus by allowing the application of abstractions to create new molecular graph rewrite rules. This is useful in modeling cellular differentiation: when a particular pattern is found in the system, the application of such an abstraction introduces new rules specializing more the behavior of particular molecular complexes.

The structure modeling the state of the system and the current set of abstractions is itself a port graph built with a node [ ] and distinct auxiliary ports called *handlers* for each node. The handler of the node [ ] is connected to the handlers of all nodes of the molecular graphs and to the handlers of the arrows of all abstractions.

Using a similar mechanism as in the CHAM, an interaction takes place in a system by heating it up. This process isolates an abstraction (or a list of abstractions) and a molecular graph for application and connect them with an application node @. A list of abstractions is defined by a new node ⟨ ⟩ which connects an abstraction and another list of abstractions, possibly empty.



All steps computing the application of abstractions to a molecular graph, including the matching and the replacement operations, are expressible using port graph transformations by considering some more auxiliary nodes and extending the reduction relation. This reduction mechanism is internalized in the calculus.

Instead of having a highly non-deterministic behaviour of molecular graph rewrite rules application, one may want to introduce some control to compose

or choose the rules to apply, possibly exploiting failure information. The notion of abstraction is powerful enough to express such control, thanks to the notions of *strategy* and *strategic rewriting* [13]. Strategies are higher-order functions that select rewriting derivations. Various strategy languages have been proposed in ELAN [10], Stratego [16], TOM [4], or Maude [15]. In a strategy language, the basic elements are the rewrite rules and the identity (id) and failure (`fail`) strategies. Based on them, strategies expressing the control can be constructed, like the sequence (`seq`), the left-biased choice (`first`), the application of a strategy only if it is successful (`try`), and the repeating strategy (`repeat`).

In the ρ_{pg}-calculus, strategies are abstractions, hence objects of the calculus. Considering also a failure node stk, we encode for instance the strategies id, `fail`, and `seq` as the following abstractions:

$$\mathsf{id} \triangleq X \Rightarrow X \qquad \mathtt{fail} \triangleq X \Rightarrow \mathsf{stk} \qquad \mathtt{seq}(S_1, S_2) \triangleq X \Rightarrow \langle S_1 \langle S_2 \rangle \rangle\ X$$

Then the strategies `first`, `try`, and `repeat` are easily defined using the above strategies and some reduction rules explicitly handling the failure node stk.

Thanks to strategies, the heating rule is reformulated based on a failure catching mechanism as follows: if $\langle S \rangle @ M$ reduces to the failure, i.e., to the stk node, then the strategy $\mathtt{try}(\mathsf{stk} \Rightarrow S\ M)$ restores the initial strategy and molecular graph subjects to reduction.

$$(\textbf{Heating}') \qquad [X\ S\ M] \longmapsto [X\ \langle \mathtt{seq}(S, \mathtt{try}(\mathsf{stk} \Rightarrow S\ M)) \rangle @ M]$$

After the application of a strategy on a molecular graph successfully takes place, a *cooling* rule, the counterpart of the heating rule, is in charge of rebuilding the state of the system by removing the no longer useful application nodes and plugging the result of the (strategic) rewriting in the environment.

The successful application of an abstraction or strategy to a molecular graph produces a new graph, built according to one chosen matching solution.[4]

At this level of definition of the calculus, the strategies are consumed by a non-failing interaction with a molecular graph. One advantage is that, since we work with multisets of port graphs, a strategy can be given a multiplicity, and each interaction between the strategy and the molecular graph consumes one occurrence of the strategy. This permits controlling the maximum number of times an interaction can take place. But sometimes, it may be suitable to have persistence of the information concerning the available abstraction and thus the persistence of a given possible interaction. In this case, the abstraction should not be consumed by the reduction. For that purpose, we define the *persistent* strategy that applies a strategy given as argument and, if successful, replicates itself. Again, we encode this strategy as an abstraction:

$$S! \triangleq X \Rightarrow \langle \mathtt{seq}(S, \mathtt{first}(\mathsf{stk} \Rightarrow \mathsf{stk}, Y \Rightarrow Y\ S!) \rangle\ X$$

[4] An alternative would be to consider a structure of all graphs corresponding to the different matching solutions. This would assume a new node for composing possible results with appropriate reduction rules considering such structures. This is not developed here.

Using the capability of strategic rewriting to generate all possible states of a system, the framework can already be used for the verification of some properties (like the presence or the absence of certain molecular graphs) as soon as such properties can be encoded as objects of the calculus. In [3] we showed how the principles of the ρ_{pg}-calculus are expressive enough for modeling systems with self-organizing and emergent properties and illustrated it on a mail delivery system.

For future work, we plan to identify conditions on abstractions for accessibility of stable states of modeled systems, or for imposing fairness on the application of abstractions, and to integrate verification techniques in the calculus. Another interesting feature worth and quite natural to be defined in the calculus represents the possibility of modifying or deleting abstractions as objects of the calculus, with application in modeling cellular dedifferentiation for instance.

References

1. Andrei, O., Kirchner, H.: A Rewriting Calculus for Multigraphs with Ports. In: Proceedings of RULE'07. (2007)
2. Andrei, O., Kirchner, H.: Graph Rewriting and Strategies for Modeling Biochemical Networks. In: Proceedings of SYNASC'07, IEEE Computer Society (2007) 407–414
3. Andrei, O., Kirchner, H.: Strategic Port Graph Rewriting for Autonomic Computing. In: Proc. of TFIT'08. (2008)
4. Balland, E., Brauner, P., Kopetz, R., Moreau, P.E., Reilles, A.: Tom: Piggybacking rewriting on java. In RTA'07. Vol. 4533 of LNCS, Springer-Verlag (2007) 36–47
5. Banâtre, J.P., Fradet, P., Radenac, Y.: A Generalized Higher-Order Chemical Computation Model. ENTCS **135**(3) (2006) 3–13
6. Banâtre, J.P., Fradet, P., Radenac, Y.: Programming Self-Organizing Systems with the Higher-Order Chemical Language. IJUC **3**(3) (2007) 161–177
7. Banâtre, J.P., Métayer, D.L.: A new computational model and its discipline of programming. Technical Report RR-566, INRIA (1986)
8. Berry, G., Boudol, G.: The Chemical Abstract Machine. TCS **96**(1) (1992) 217–248
9. Bertolissi, C., Baldan, P., Cirstea, H., Kirchner, C.: A Rewriting Calculus for Cyclic Higher-Order Term Graphs. ENTCS **127**(5) (2005) 21–41
10. Borovanský, P., Kirchner, C., Kirchner, H., Ringeissen, C.: Rewriting with strategies in ELAN: a functional semantics. IJFCS **12**(1) (2001) 69–98
11. Cirstea, H., Kirchner, C.: The rewriting calculus - Part I and II. Logic Journal of the IGPL **9**(3) (2001) 427—498
12. Danos, V., Laneve, C.: Formal molecular biology. TCS **325**(1) (2004) 69–110
13. C. Kirchner, F. Kirchner, and H. Kirchner. Strategic computations and deductions. Festchrift in honor of Peter Andrews, LNCS, Springer-Verlag, 2008.
14. Laneve, C., Tarissan, F.: A simple calculus for proteins and cells. ENTCS **171**(2) (2007) 139–154
15. Martí-Oliet, N., Meseguer, J., Verdejo, A.: A Rewriting Semantics for Maude Strategies. In: Proc. of WRLA'08. (2008)
16. Visser, E.: Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5. In RTA'01. Vol. 2051 of LNCS, Springer-Verlag (2001) 357–361